Shared Understanding of the User Experience: A Case Study of Collaboration Between Developers and Designers

Jukka-Pekka Salo

Helsinki May 2020

UNIVERSITY OF HELSINKI
MSc Thesis
Master's Programme in Computer Science

HELSINGIN YLIOPISTO − HELSINGFORS UNIVERSITET – UNIVERSITY OF HELSINKI

Tiedekunta/Osasto − Fakultet/Sektion –Faculty/Section	Koulutusohjelma − Studieprogram − Study Programme
Faculty of Science	Department of Computer Science

Tekijä − Författare − Author

Jukka-Pekka Salo

Työn nimi − Arbetets titel − Title

Shared Understanding of the User Experience:
A Case Study of Collaboration Between Developers and Designers

Ohjaajat − Handledare − Supervisors

Petri Kettunen

Työn laji − Arbetets art − Level	Aika − Datum − Month and year	Sivumäärä − Sidoantal − Number of pages
Master's Thesis	May 2020	55 pages

Tiivistelmä − Referat − Abstract

User experience has become vital for many software development projects but the software development methods and tools are not originally intended for it. Moreover, software development is fundamentally complex and an increasingly social profession. This shift towards designing for user experience as a diverse group has brought new challenges to software development.

The objective of this study is to find how developers and designers form a shared understanding of the software system UX under development. Central theme are the activities of UX work: what are the methods in use (e.g. User-Centered Design, Agile) and how do they work in practice, that is, what kind of information developers and designers share and what kind of artifacts do they produce in collaboration. This study answers two research questions: (RQ1) How do developers and designers form a shared understanding of the software system UX under development; and (RQ2) What are the artifacts utilized in their collaboration.

To answer the research questions, a single case study research was conducted by interviewing the employees of a Finnish startup company. The company develops enterprise resource planning software (ERP) for rental businesses.

The results show that shared understanding of the UX is achieved with UX activities throughout the system's lifecycle where the user participation is required from the beginning of new software development. Furthermore, the artifacts in combination with developers' participation in some of the UX activities will convey the design intent to the implemented software.

ACM Computing Classification System (CCS)
Human-centered computing → Interaction design → Interaction design process and methods → User centered design

Software and its engineering → Software creation and management → Collaboration in software development → Programming teams

Avainsanat – Nyckelord − Keywords

user-centered design, user experience, collaborative software engineering

Säilytyspaikka − Förvaringställe − Where deposited

-

Muita tietoja − Övriga uppgifter − Additional information

Thesis for the Software Systems study track

Contents

1 Introduction

Software engineering is inherently a social profession and implementing software depends on collaboration between group members. Requirements do not necessarily originate from cutting edge high technology but from customer's business aspirations or user's needs, which brings a new set of problems to be solved—problems that are sociological, not technical. (DeMarco & Lister, 2013, p. 4; Fagerholm, 2015; Fox et al., 2008; Whitehead, 2007.) Furthermore, the goal for many projects is to develop complex interactive software systems and design for user experience (UX). Responding to the changed demands for system quality attributes by incorporating design thinking, such as User-Centered Design (UCD), into iterative development is applicable to these circumstances but clear guidelines have not yet surfaced. (Fox et al., 2008; Kuusinen, 2015; Larusdottir et al., 2017; Magües et al., 2016; Ralph, 2015.)

Software systems are built by people. In the context of UCD the people who form a software development group may come from various backgrounds. The cross-functional interdisciplinary group is therefore a heterogeneous set of people and initially they may have different—atomistic instead of holistic—perspective of what is important in the system under development (Gothelf & Seiden, 2016, p. 41; Holtzblatt, 2016, p. 82). Software system the people are about to build is more than the sum of its parts, it has emergent properties, so the group that is implementing the system has to form a shared understanding to achieve conceptual integrity and ultimately a quality software and desired UX (Kuusinen, 2015).

This study examines how do developers and designers form a shared understanding of the software system UX under development. While "developer" and "designer" are separate professions, the boundaries between the roles have started to blur. Collaboration is thus a central concept in this study and it is viewed as communication and producing artifacts between developers and designers. Communication and the artifacts developers and designers produce are means to form a shared understanding of the system UX among the group. From these premises collaboration has two purposes: (1) problem solving; and (2) information sharing. In the collaborative work, the group

switches autonomously between the activities of exploring the problem and solution space throughout the process and produces artifacts to represent the design intent.

1.1 Research Questions

The objective of this study is to examine how UX is acknowledged in product development—how a group of people is able to solve relevant problems in terms of the system's lifecycle and evolution so that they are continuously able to achieve desired UX. This study focuses on iterative software development, and how well it is applied in the context of user needs and requirements elicitation. The central theme is collaboration between developers and designers: what are the iterative methods (e.g. UCD, Agile), how do developers and designers apply them in practice and how do they acknowledge the term "user experience".

The contextual framework of this study is collaborative software engineering, design thinking, Human-Computer Interaction (HCI) and UX. The main research problem addressed in this study is: What kind of knowledge is shared between developers and designers? Specifically, this study aims to answer the following research questions:

RQ1 How do developers and designers form a shared understanding of the software system UX under development?

RQ2 What are the artifacts utilized in their collaboration?

1.2 Thesis Structure

This thesis is organized as follows. Section 2 presents the central constructs and relevant previous work which is later compared with the findings. Section 3 presents the research design, rationale for case study as a research method and provides definition of the case. It will then describe how interviews as data collection method and Grounded Theory as data analysis method are used in this study. Section 4 presents the analysis and results. In Section 5, the synthesis of the results is evaluated in the light of related work. Finally, Section 6 concludes this thesis and outlines future work.

2 Previous Work

2.1 User Experience Design

> *"Design is a conscious and intuitive effort to impose meaningful order." (Papanek, 1985, p. 4)*

Designing or design thinking is an act where an agent (individual or group) specifies or builds an object to accomplish a goal in a given context with constraints. Goal is considered accomplished when the agent is able to come up with a single satisfactory solution with an appropriate result. (Cross, 2011, p. 28; Ralph, 2015.) Design methods in general yield qualitative data about the context and focus on people's activities and understanding their needs (Norman, 2013, p. 7, 2013, pp. 224–225). Proposed design solution is not necessarily best even though it would satisfy all the requirements because many ideas could achieve the same functionality and, in theory, one of them is a "perfect fit". For example, even though a system's quality would be ensured and it would satisfy the requirements on schedule and on budget, the user interface (UI) could be inconsistent and difficult to use. In other words, while the problem is successfully solved, the problem setting is ignored. (Fairbanks, 2010; Lim et al., 2008; Norman, 1998, p. 53; J. Patton, 2014, p. xix, 2002; Schön, 1984, p. 40; Sohaib & Khan, 2010.)

Purao et al. (2002, as cited in Ralph, 2015) defined problem and solution space as metaphors for how requirements are interpreted and what are the specific solutions. In the non-academic literature they are referred to as "designing the right thing" and "designing the thing right", a somewhat reverse version of software verification and validation (cf. Boehm, 1984). Designing starts by rigorously defining the initially ambiguous problem before finding a solution. The problem definition is a manifestation of the most important qualities in its simplest form, preserving the understanding of the whole. This troublesome gap between the problem and possible solutions becomes more apparent later in this section when the UCD and Agile processes are put into comparison. (DeMarco & Lister, 2013, pp. 19–23; Lim et al., 2008; Norman, 2013, p. 219, 2013, p. 234; Ries, 2011, p. 37.)

To define the core problem, people involved in the design need to discard many great insights if they are not suitable for the users or for the system's holistic behavior. This can be done, for example, by sketching the whole concept first and preserving the good parts only. (Cross, 2011, p. 51; Heath & Heath, 2007, p. 28; J. Patton, 2014, p. xix.) The process of detecting the essence also contributes to the understanding of what needs to be adjusted if the idea is wrong (J. Patton, 2002). On the other hand, Rowe (1987, as cited in Cross, 2011, p. 22) observed that people prefer trying to make the initial idea work instead of starting over when they face critical problems. These sort of "failures" can in the right circumstances be considered as learning experiences in a journey towards building usable systems (Norman, 2013, p. 229).

Verifiable definition of the core problem contributes to the system's conceptual integrity which can be interpreted that it is better to focus on a few coordinated design ideas and to exclude inconsistent features and improvements (Brooks, 1995, p. 42). Although the design is not likely complete or correct at the first time, up-front design improves product consistency and builds group's shared understanding (DeMarco & Lister, 2013, p. 8; Ferreira et al., 2007, as cited in Kuusinen et al., 2012; Ungar & White, 2008, as cited in Sohaib & Khan, 2010). The need for consistency is omnipresent throughout the implementation: change and iteration is inevitable because context is expected to be unstable (Ralph, 2015). Additionally, according to the law of continuing change on product evolution (Lehman, 1979): the system must be continually changed or it becomes progressively less useful—or gradually obsolete from a purely technical perspective. If the system has to change, every member of the group has to change too; all the people must constantly converge on a single design (Norman, 2013, p. 149; Whitehead, 2007).

Agile practitioners have generally avoided the up-front design phase and its repercussions—mainly the inability to change it later—can be traced back to the era of waterfall development. Regardless, the implementation does not usually start before some initial design is complete and complex design problems may require long up-front design. (Cross, 2011, p. 4; Kuusinen et al., 2012; J. Patton, 2002; Sohaib & Khan, 2010, Ferreira et al., 2007, as cited in 2010; Sy, 2007.) In this sense, User-Centered Design (UCD) prior to more traditional Agile development is suitable: UCD advocate for

up-front design phase, such as contextual inquiry and low-fidelity (lo-fi) prototyping in order to understand users' needs. (Kuusinen, 2015; see Magües et al., 2016 for methods; Sohaib & Khan, 2010.)

User-Centered Design (UCD) is an iterative method where the user involvement is the focal point of the process. UCD consists of four distinct phases: (1) understand the context; (2) specify the users' requirements; (3) create design solutions; and (4) evaluate designs. There can also be an additional planning phase before the iterative work begins. (Sohaib & Khan, 2010: see Figure 1.) UCD is an exploratory process, not a specification for, but a validation of a solution. Design activities performed in the aforementioned phases form an intertwined and flexible relationship between problem settings and problem solving where the problem is incrementally reframed and represented in the intermediate artifacts. (Cross, 2011, p. 8, 2011, p. 25; Lim et al., 2008; Ralph, 2015; Salah et al., 2014.)

However, the actual software development processes are prone to oversimplify and over-rationalize reality to some extent and detach lifecycle phases—such as design and development—into sequential, cohesive and mutually exclusive activities (Ralph, 2015; Sy, 2007). Despite the iterative methods, by definition, defer formation of rigid design specifications, the activities are complicated and variations of iterative development processes might omit some of them altogether even though they should continue throughout the project, for example in the form of UCD (Norman, 2013, p. 234; J. Patton, 2002; Sy, 2007). In practice, the UX design phase is usually time-boxed thus not deriving validation of a solution. (Fox et al., 2008; Kuusinen et al., 2012.)

The remainder of this section is dedicated to propose a view towards user experience in this study. Acknowledging the plethora of definition attempts for—and attempts for not defining—user experience or its constructs, this study approaches user experience as a phenomenon the human possesses (qualia), with or without a presence of computer interaction counterpart. According to the International Organization for Standardization (2019), user experience is defined as "user's perceptions and responses that result from the use and/or anticipated use of a system, product or service". Marc Hassenzahl writes in his book *Experience Design: Technology for All The Right Reasons* that user experience is quite similar to a "generic" experience (Hassenzahl & Carroll, 2010). A

difference between them, Hassenzahl continues, is that in user experience the person is assumed to interact with a computer at some point, although experiencing occurs before, during and after the interaction. User experience is similarly distinguished from sheer experience in a survey for UX researchers and practitioners about defining UX (Law et al., 2009). In the survey, the respondents agreed on the idea that "UX is based on how a person perceives the characteristics of an artefact, but not on the characteristics per se" (Law et al., 2009). Respondents also agreed that the user experience is subjective but disagreed with the uniqueness of it in terms of measurability. (Bargas-Avila & Hornbæk, 2011.)

Furthermore, the view of UX work in this study is along the lines of how Kuusinen (2015) has defined it:

> By UX work, we refer to activities that aim at developing software that is usable, fulfills user needs, and provides desired UX. The work can be conducted by any project member; we do not limit the definition of UX work to a certain role (e.g. UX specialist). (p. 3)

UX work seems to mean different things in the context of Agile and in UCD. While the certain UX specialist role may be omitted in either of them, UCD as a method implicitly demands that some members are experienced in UX whereas Agile does not. The latter involves the customer and other stakeholders to the process as a feedback and requirements provider. The stakeholders, however, may not be real users of the system under development but experts in the constraints of the domain. They may effectively lack necessary knowledge to "require" usability and Agile as a method positions itself more to focus on the functionality than on usability. Still, the needs of the end users may originate from the stakeholders who likely have a lot of domain knowledge and business aspirations but not necessarily the needs and experiences of the user nor the user perspective held by a dedicated UX specialist. Stakeholders will likely identify the symptoms, which may be the starting point to find out the root cause. (Dhandapani, 2015; Larusdottir et al., 2017; Magües et al., 2016; J. Patton, 2014, p. xviii; Sohaib & Khan, 2010.)

Coalescing the perspectives of what user experience is and how the UX work is conducted leads this study to an eloquent proclamation from Benyon (2013) about not designing an experience, rather designing *for* experience: "Experiences, therefore, cannot really be designed. Designers can design for experience, but it is individuals and groups who have the experience" (p. 94).

2.2 Collaboration in Software Engineering

> *"Good people, with good skills and good judgment, are what make projects work."* (Boehm, 1991)

Software is made by a group, an independent and entitative agent. Most of the applied computer science, software engineering, includes people with diverse skills from different areas, such as developers, designers, clients and users which form an independent agent, a cohesive software development group. (Helkama, 2015, pp. 266–268; Ralph, 2015.) As stated in Section 2.1, the development lifecycle phases are not linear, separate entities but activities that the independent agent is working on and moves freely between them (Ralph, 2015).

This study approaches software engineering collaboration with a focus on creating artifacts which generate new shared understanding and provide shared understanding to others when they are finished (J. Patton, 2002; Sy, 2007; Whitehead, 2007). Creating artifacts can itself be a collaborative activity and the artifacts are a means to an end, a transient way to create the system (Sy, 2007; Whitehead, 2007). Collaboration in the context of this study is essentially activities and transitions between them performed by a cohesive group (Ralph, 2015).

Software engineering has taken a move towards a social and diverse profession. Complex problems, where the solution is not known a priori, such as achieving a desired UX, exceeds the capabilities of a single person and thus depends on the synergy between the members of a group to solve them (Brooks, 1995; Kuusinen, 2015; Ferreira et al., 2011, as cited in Larusdottir et al., 2017; Ralph, 2015). According to Bales (1953, as cited in Brown, 2001, pp. 40–41), a group exists to achieve a task (i.e. a desired UX) while maintaining relationships. The progress towards achieving the task emphasizes

how the group interacts, that is, how people align with the task and communicate about it (DeMarco & Lister, 2013, p. 136; Helkama, 2015). In practice, if the task, or problem, is split into multiple sub-problems, it is imperative that the people understand the high-level design intent to preserve the holistic view of the system (Kuusinen et al., 2012; Sy, 2007). This was found especially important in a case study about the integration of UCD and Agile practices (Kollman et al, 2009, as cited in Sohaib & Khan, 2010).

DeMarco & Lister (2013, p. 5) write about the sociological aspects of software engineering and draw a distinction between software engineering and computer science: majority of software engineering is not high-tech, it is applying the research done in high-tech. Furthermore, the demands for a typical software engineering project originate from understanding the customer's business needs and user's needs rather than understanding the intricacies of high technology and exploiting the potential of it. Regardless, some organizational cultures emphasize technology, for example, by arranging people into departments by functional area, which might come at the expense of reduced communication and therefore reduced user experience as argued in the previous paragraph. (Kuusinen, 2015; Norman, 1998, p. 201; J. Patton, 2014, p. xviii; Schön, 1984, p. 34, 1984, pp. 44–45; Sohaib & Khan, 2010.) In this kind of context, the system and technologies dominate the intended user activities, and people may try to classify and simplify the problems so that they are able to solve them with available techniques. Even though problem solving is technical, problem setting is not as technical a problem as the aforementioned circumstances might suggest. Collaborative brainstorming and discussion are examples of efficient and non-technical activities to frame the problem, elicitate the requirements and increase knowledge and confidence (Norman, 1998, p. 56; J. Patton, 2002; Schön, 1984, p. 40, 1984, p. 41, 1984, pp. 44–45).

Software development group members (i.e. team) usually depend on the actions of others and the way they collaborate can impact the project significantly because the way of working is in part formed (and changed) as a result of the activities of the group. (Lewin, 1948, as cited in Brown, 2001, p. 35, Sherif, 1936, as cited in 2001, p. 59; Whitehead, 2007.) Iterative software development, such as UCD or Agile development,

is a bottom-up human-centered approach, and close collaboration is evidently important (Miller, 2005, as cited in Kuusinen et al., 2012; Mistrík et al., 2010, p. 7). In one case study (Sy, 2007), the design and implementation progressed on separate tracks but in reality, the developers and designers needed to communicate daily to exchange bi-directional information. Collaboration is both vertical (i.e. with managers) and horizontal (i.e. with peers) so traditional hierarchical structure and top-down processes are becoming less suitable for product development. (DeMarco & Lister, 2013, pp. 176–177; Norman, 1998, p. 215; Ralph, 2015.)

In some situations, the design agent will hand off its results as a specification to other disciplines, which consequently revisit the earlier discussions to fit it to their needs and potentially cause changes to the scope (e.g. Ferreira et al., 2010; Kuusinen et al., 2012; J. Patton, 2002; Sy, 2007). The diverse opinions of the people—especially developers—involved in a project should be taken into consideration in the design because it is developers' task to transform the design intent into reality without guesswork and without subsequent changes which tend to weaken the cohesion of the system (Kuusinen et al., 2012; Norman, 2013, pp. 241–242). This—linear but essentially a top-down—handoff procedure can lead to a lot of non-deployed design or deviate the actual implementation and the design intent into separate tracks, especially when there is a rather long delay between them (Kuusinen, 2015; Sy, 2007). Moreover, if the implementation is the last phase in the lifecycle, it may require the developers to conform to the "finished" design they receive (Fox et al., 2008; Kuusinen et al., 2012). Although, when it works, that is, the design barely precedes implementation and artifacts convey the information properly, it can be a key to desired UX and for providing early feedback (Meszaros, 2006, as cited in Fox et al., 2008; Kuusinen et al., 2012; Sy, 2007).

Any kind of communication is imperative for the people to converge on a single software design (Whitehead, 2007). As Conway's law states about software design, "Organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations" (Conway, 1968). Collaboration is anything from unstructured and informal communication to structured and formal use of artifacts. Knowledge is often exchanged in a dialectic process with

natural language which is expressive but ambiguous, and many technical tools support it. The conversations are usually not formally structured even though they—counterintuitively—concern the development of a formal and deterministic system, whereas artifacts provide structure and promote shared understanding and can act as a reference for previous discussions (Helkama, 2015; J. Patton, 2002; Ralph, 2015; Sy, 2007; Whitehead, 2007). Additionally, "documentation does not reduce the amount of communication required" (Ralph, 2015), and even if it would, some people prefer conversation over written material (Sy, 2007). People can articulate their own understanding of the system with a combination of artifacts and natural language and new members are able to assimilate the vocabulary (J. Patton, 2002; Sy, 2007; Whitehead, 2007). Based on the presented argumentation about collaboration and communication as both handicap and advantage towards desired UX, also the individual communication skills obviously matter; subpar or confusing UX may be partially caused by lack of communication (DeMarco & Lister, 2013, pp. 105–106; Kuusinen et al., 2012; Norman, 2013, p. 73, 2013, p. 137).

2.3 Shared Understanding of the Software System UX Under Development

> *"Our knowing is in our action"* (Schön, 1984, p. 49)

Professionals usually know more they can or have to say, they have practical experience and possess knowledge that is tacit (Schön, 1984, p. viii). Thus, people may have tacit or idiosyncratic knowledge about UX but the organization may lack an overarching and shared knowledge. The capability of the organization to facilitate the shared understanding is in part determined by the culture and (covert) values (Ferreira et al., 2010) and in part by the level of organization's maturity (see Salah et al., 2014). Schein's model of organizational culture aligns underlying assumptions at the bottom of a triangle, followed by values below the tangible artifacts. The model presents that the values and assumptions will scope and serve as a basis for people's expected behavior and the way of knowing and shape the use of tools and methods. (Schein, 2010, as cited in Bock, 2015, pp. 29–30; Schein, 1988, as cited in Christensen, 2016, p. 169; Larusdottir et al., 2017.) Maturity Models on the other hand are models to assess the

organization's current maturity stage and evaluate the logical actions to level up to the next stage. Nielsen Corporate Usability Maturity Model is an eight-stage model specifically for UX (Salah et al., 2014). Furthermore, the actual software development method the professionals practice (i.e. identified as Agile by the professionals) may differ from the prescribed method. Mathiassen (1998, as cited in J. Patton, 2002) observed that professionals do not always follow methods although deliberately chosen methods increase the odds of providing a better UX. (Larusdottir et al., 2017; J. Patton, 2002; Ralph, 2015; Ralph & Narros, 2013.) A review about user-centered systems design in Agile reported the challenges of achieving basic shared values among design and development disciplines. According to the review, there is a predisposition towards systems perspective in software engineering. It requires the UX professionals to justify their methods and assimilate developers' nomenclature to be able to integrate into the already-existing unilateral structure of people and the development process and ceremonies they adhere to. (Larusdottir et al., 2017.)

This section started with the organizational and methodological dimensions for shared understanding but will now move towards individual and system level. Knowledge and shared understanding in the context of this study refer to more subjective and intangible beliefs and ideas about the design (J. Patton, 2014, p. xxxiv; Ralph, 2015). Peoples' understanding evolves, and knowledge acquiring happens throughout the process. Additionally, acquiring knowledge from the user is easier than to synthesize it into a shared understanding. In this sense, the shared understanding in the form of a finished design solution is a composition of iterative sketching and interaction with intermediate artifacts. (Holtzblatt, 2016, p. 81; Lim et al., 2008; Ralph, 2015.)

Conceptual model is at the essence of UX, both on the user's side and on the system's side. On the user's side it consists of the series of actions a person has to perform (as a user of an interactive system) in order to achieve her goal or fulfill her need. A goal can be a mundane job to be done out of necessity or a more emotion-evoking hedonistic achievement. On the system's side, the implementation is the representation of the thinking of the people, the present state of their knowledge about the users' conceptual model. Additionally, the preferable way for them to communicate the system's behavior to the user is via the system's user interface, not through a manual or instructions for

example (Norman, 1998, p. 179). Discrepancy between the system's behavior and the user's conceptual model or a failure to communicate it via the system will appear as the user's difficulties in understanding and using the system. (Norman, 1998, p. 173, 2013, pp. 31–32.) Task analysis and simulation-based design are available methods to elicit a user's conceptual model and evaluate the system against it whereas affordances and signifiers in the UI are a means to communicate the system's behavior to the user (e.g. Lane et al., 2006; Norman, 2013, pp. 13–19; Stanton, 2006). However, in the context of a UX, a conceptual model is orthogonal to usability: it doesn't matter if the usability is poor or great because the user will form a conceptual model in any case and will eventually learn to use the UI if she decides to continue achieving her goal (see "habituation"). Hence, the conceptual model should make a clear distinction between satisficing and satisfying UX—from the user achieving the goal accidentally to achieving it with a sense of control. Conceptual models are especially important with software systems due to their intangible and interactive nature (Norman, 1998, p. 173).

Both of the dimensions are important for a group's shared understanding. Additionally, trade-offs, such as balancing between externally imposed constraints like the budget or time and the quality, are part of software development (Norman, 1998, p. 211). The understanding among group members may be inconsistent, especially when the goals are complex, unclear, unknown or in conflict (Cross, 2011, pp. 19–20; Ralph, 2015). Unlike technical aspects, sociological aspects aren't exact by nature. Therefore the espoused organizational values should exist to support and create space for (1) the collaboration of the autonomous group and for (2) the shared understanding on the system and individual level. Designing begins from these premises by exploring the context, not alone, but as a group with a goal to achieve a consensus—intersubjectivity—among participants. This mutual iterative refinement is central to design. (Bucciarelli, 1994, as cited in Cross, 2011, pp. 19–20; Ralph, 2015; Sy, 2007.) Designer can communicate the conceptual model through the prototype (Lim et al., 2008). A study showed that when two groups interacted with different prototypes, they formed significantly different conceptual model of the system and opinions about the functionality (Lim et al., 2008).

2.4 Representing Design Intent in the Artifacts

Intermediate artifacts are part of software engineering. They are public so other people can see them. (Cross, 2011, p. 20; Ralph, 2015.) People create artifacts to help them to get their work done and artifacts can express the knowledge that led to its creation (Beyer & Holtzblatt, 1998, pp. 102–103; Fairbanks, 2010, p. 193). Artifacts reveal concepts for people who work with them and they can be used to capture emergent design patterns (Beyer & Holtzblatt, 1998, pp. 102–103; Lim et al., 2008). The artifacts help people to form shared understanding and to find a common ground by solving conflicting interests or trade-offs, especially when the emergent properties are not explicitly represented in the requirements. (J. Patton, 2014, p. xxxiii; Whitehead, 2007.)

Meticulously chosen set of artifacts focus thinking and discussion between problem and solution that would otherwise be obscure, aimless and eventually forgotten. Some artifacts are formally required by the process, for example, to protect from knowledge loss and miscommunication, but, in practice, the artifacts in use depend on the circumstances. (Brooks, 1995, p. 108; Cross, 2011, p. 12; J. Patton, 2014, pp. 6–7; Ralph, 2015; Whitehead, 2007.) Initially, the value of an intermediate artifact, especially digital tools, may look rather enticing and beneficial, but the total number of artifacts should be kept minimal to avoid unnecessary overhead and maintenance (DeMarco & Lister, 2013, p. 225; Whitehead, 2007). A key insight from one case study (Sy, 2007) was that nobody reads the documentation—or uses the artifacts—but the group, thus they should be produced with the target audience in mind with a goal to communicate the ideas. A principle from Manifesto for Agile Software Development (2001) recommends producing “working software over comprehensive documentation”. The differences between Agile and UCD is, however, that the former aims to minimize documentation, whereas the latter aims to produce **good** documentation as is the case with converging on a single design. (Agile Alliance, 2001; Larusdottir et al., 2017; Salah et al., 2014; Sohaib & Khan, 2010.)

Even though design and development happens largely inside a person’s mind, only internal mental activities are not enough, but developers and designers alike interact

with external representations (Cross, 2011, p. 12, 2011, p. 71; Lim et al., 2008; Seaman, 1999). People externalize their thinking, amplify their intelligence with artifacts where the thinking is done with the representations (Lim et al., 2008; Norman, 1994, p. 51; J. Patton, 2014, p. xxxiii). People do things in an idiosyncratic manner, so external representation is an objective way to communicate the design and to let others express their thoughts about it (see boundary objects) (Beyer & Holtzblatt, 1998, p. 139; Schön, 1987, as cited in Lim et al., 2008). Externalization is a way to "predict the future", that is to evaluate and suggest how the design might be used before the system is implemented and made available for users (Gedenryd, 1998, as cited in Cross, 2011, p. 28, 2011, p. 47; Larusdottir et al., 2017; Lim et al., 2008).

A prototype is a fundamental artifact in UCD—and increasingly in software development in general (Memmel et al., 2007, as cited in Sohaib & Khan, 2010). Prototype is a tool for reflecting and traversing a design space, manifesting and reifying design ideas and accumulating knowledge towards the final design (Lim et al., 2008). Prototype is appropriate for designers' "intuitive" reasoning: design process does not follow the laws of nature but designer's own thoughts and actions, free will and creativity (Cross, 2011, p. 27; Lim et al., 2008; Ralph, 2015). Prototype's use and purpose can be (1) evaluation and testing; (2) understanding UX; (3) idea generation; and (4) communication (Lim et al., 2008). Literature exists to support the notion that a sensible way to determine feasibility and behavior of a design is to build a prototype (see: Norman, 2013, p. 227; J. Patton, 2002; Ralph, 2015). Iterative development encourages rapid prototyping, such as low-fidelity (lo-fi) paper prototypes which are suitable for idea generation (Lim et al., 2008; Norman, 1998, p. 216). Prototyping may give the impression that there is no real progress at first but—speaking in terms of problem and solution space—it is better to frame and solve the problems with inexpensive and malleable prototypes than with the actual software system: disposable artifacts have little or none sunk costs. While it is easy to change a design when it is still in prototype form, sometimes fast answers are preferred over correct answers or there is otherwise no time for immediate testing and re-assessing the problem but it is postponed opportunistically to an imaginary next version in the unforeseeable future. (Larusdottir et al., 2017; Norman, 1998, pp. 193–195, 1998, p. 216; Sy, 2007.). A prototype is not the final product—it should not be used for proving solutions but for exploring and

discovering problems (Lim et al., 2008; Norman, 1994, p. 49). Yet there is a lack of knowledge about this fundamental nature of prototype and the benefits may be missed with incorrect use or understanding (Lim et al., 2008), such as without the attitude towards failures as learning experiences (see Section 2.1). The representational form dominates how the prototype will convey the properties and conceptual model of the design, and will impact how the user of the prototype, such as developer, experiences and interprets the final design (Lim et al., 2008).

In terms of design intent, requirements are difficult to "get right" and many projects lack explicit requirements for developers to be able to implement the system (Kuusinen et al., 2012; Norman, 2013, p. 229; Ralph, 2015). In Agile, work is split into smaller chunks, traditionally into a requirements list in a backlog. While smaller pieces of things to be done have obvious benefits, the thought process on a feature-level implementation as opposed to thinking about the system as a whole is different: requirements list may exist in a vacuum and do not convey the meaning and interconnected nature of features nor the logic and data model beneath (Norman, 1998, p. 207; J. Patton, 2014, p. xi, 2014, p. xxiii, 2014, p. 1, 2014, pp. 3–6). Prototype as the manifestation of design intent may be misinterpreted as the final solution and used to identify and satisfy requirements or estimates. User stories, on the other hand seem to be suitable for this purpose (Fox et al., 2008) Eliciting specific requirements should preferably be postponed until enough is known via iterative testing and rapid prototyping but early enough so that the implementation does not start for undesigned features. (Lim et al., 2008; Norman, 2013, p. 235; J. Patton, 2014, p. xviii, 2002; Sy, 2007.)

Artifacts are abstractions which pose a challenge to know what should be abstracted—or what is the level of fidelity. Artifacts should solve the "main problem", that is, they are not supposed to consider peripheral details or mundane "sub-problems", such as making things "pixel-perfect". (Cross, 2011, p. 18; Gothelf & Seiden, 2016, p. 58; Lim et al., 2008; Norman, 1994, p. 49, 1994, p. 52.) An abstraction that is suitable for the task increases understanding and allows extracting knowledge about specific aspects without distraction (Lim et al., 2008; Norman, 1994, p. 49, 1994, p. 52). Abstraction is "perfect", not when "'there is nothing left to add, but when there is

nothing left to take away'" (Saint-Exupéry, 1939, as cited in Heath & Heath, 2007, p. 28). It is done in an iterative manner, and, according to the economic principle, in a simple and efficient way to make the design idea measurable (Lim et al., 2008; Sy, 2007). An abstraction is incomplete in the sense that it does not explore every aspect of the design (Lim et al., 2008). For example, a feature represented in the abstraction, like the ability to add an item, generally means that its counterpart is needed as well, the ability to remove it too, but may be omitted from the abstraction. Even though "things not represented fall in importance" (Norman, 1994, p. 52), the strength is in its incompleteness: an abstraction allows to explore the design idea without implementing a fully working system (Lim et al., 2008). However, an abstraction can't be too ambiguous, too… abstract (J. Patton, 2002). People may interpret abstraction in different ways or they may not understand or remember the meaning of abstraction if it is not adequately concrete. Elaborate and abstract artifacts do not necessarily help to fix a problem if other group members do not understand them, but may increase miscommunication and difficulties in coordination. (Heath & Heath, 2007, p. 100, 2007, pp. 114–115.) That is to say, they lack a shared conceptual model.

Finally, the actual software system (Brooks, 1995, pp. 4–6) is a solution to a problem (Fairbanks, 2010, p. 193), a representation of the design intent (Cross, 2011, p. 47). The whole is greater than the sum of its parts—the system exhibits emergent behavior not evident from its constituent parts (Holland, 1992, as cited in Ralph, 2015). This behavior and the qualities emerge from architecture and software design (Fairbanks, 2010, p. 161) and, to some extent, without conscious intention (Cross, 2011, p. 11). It is the result of collaboration between multiple disciplines.

3 Research Design

According to Yin (2017, p. 1), the (first part of a twofold) definition for a case study is "an empirical method that investigates a contemporary phenomenon (the 'case') in depth and within its real-world context, especially when the boundaries between phenomenon and context may not be clearly evident". Following this definition, Runeson & Höst (2008) state that "case study is a suitable research methodology for software engineering research". However, software engineering and computer science are different from social sciences where case study is an established method. Software engineering, for example, focuses on system development rather than its usage and the environment is man-made instead of natural. (Runeson et al., 2012.) Runeson et al. (2012) defined case study in software engineering as

> an empirical enquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified. (p. 12)

When conducting a case study, the unit of analysis has to be clearly stated: it determines the type of the case study (multiple or single) and also the context which more or less affects the outcome. Since the context is an indispensable part of a case study, it should be explicitly stated in the text so that the reader can interpret how it affects the results. It is somewhat obvious that the researcher should select a case which yields information-rich data, but there might be some external limitations that affect the selection. (see M. Q. Patton, 2001.)

This study is an exploratory holistic single case study (the case is studied as a whole) (see Yin 2003, as cited in Runeson & Höst, 2008; Yin, 2017, p. 48). This type of study has a single unit of analysis, that is, only one case (project) is studied. M. Q. Patton (2001) further categorised the unit of analysis into focus areas (people, structure, geography, activity and time). According to the categories, this study focuses on people and temporal aspects: how each individual makes sense of the UX, and what was done

in a given time within the project's lifecycle, during the conception phase and iterations in particular.

3.1 Case Description

This section provides the definition and boundaries of the studied case (see Yin, 2017). This study will investigate how developers and designers form a shared understanding of the software system UX under development (RQ1) and what are the artifacts utilized in their collaboration (RQ2). The study was executed by interviewing employees in a Finnish startup company of 10 people that builds enterprise resource planning (ERP) software for rental businesses (see also "new product development"). Table 3.1 lists an overview of the company. The software has two disparate interfaces for three types of users: (1) ERP for rental shop owners and staff members; and (2) self-service check-in for their consumers. The customers (rental shop owners) are mostly based in Finland. The rationale for selecting the company for the study was convenience and that it was considered as an information-rich case, although they did not use quantitative UX measures. The interviews were conducted during one week in August 2019. Seven employees were interviewed, however, only five interviews were analyzed in depth providing saturated results. The details of the interviews and participant backgrounds are in Table 3.2.

Table 3.1: Unit of analysis

Product	Market	Established	Employees	Developers	UX-specialists / Designers	Users
Rental software: ERP, SaaS, Self-service check-in	B2B, B2B2C	Spring 2018	10	4	3	Shop owners Staff members Consumers

The company has 3 employees who self-identified as a UX specialist or designer compared to the 4 developers (Table 3.1), and the participants rated their own UX experience between 3-4 in a numeric rating scale of 1-5 (Table 3.2). Furthermore, some participants have overlap between the roles and contribute occasionally to both design

and development. The company has smaller “design” and “development” teams which have their own pace for working but the whole company is aligned under a single iterative release cycle and this thesis will refer to all the employees of the company as “team” and the aforementioned cycle as “process”. Consequently, the workflows followed by design and development teams are referred to as development workflow and design workflow. The team is co-located.

Table 3.2: Participants

Participant	Primary role	How long in the project	Work experience	Field of education	Own experience in UX (1-5)	Interview time of day
1	Design Leader	4-5 months	10 years	Media Technology	4	Morning
2	Tech Lead	~ 12 months	3 years	ICT Innovation	3	Afternoon
3	UX Designer	14 months	3 years	Information networks	3/4	Afternoon
4	CEO	18 months	3 years	Information networks	4	Afternoon
5	Design intern	3 months	2,5 years	BBA	2	Afternoon
6	Developer	3 months	3 years	Computer Science	3	Afternoon
7	Developer	3,5 months	5 months	Information technology	2	Afternoon

The company was established in Spring 2018 and it consisted of three founders (participants 2, 3 and 4) until the Spring 2019. The company’s software development so far can be roughly divided into two phases: (1) the conception phase (Spring 2018 - Spring 2019); and (2) post-conception phase (Spring 2019 onwards). The company consisted of the aforementioned participants in the first phase and grew into a size of ten during the Spring-Summer 2019. The company had customers and a steady revenue stream already in the first phase.

3.2 Methods

This case study uses semi-structured interviews as a data collection method. Interviewing is, according to Lethbridge (2005, as cited in Runeson & Höst, 2008), first degree (direct) data collection method within three levels.

As Seaman described qualitative study (1999), "all data is potentially useful and the usefulness of a particular piece of data often is not known until long after it is collected". This, in combination with the view of UX in this study (see Section 2.1) and with the premise that software development group is generally diverse, means that the participants do not need to be experts in UX, rather this study is interested in all perspectives and experiences on UX as well as UX as a defined objective of the project. Regardless, 4 out of 7 participants had either previous experience about UX work or were working on UX during the time of interviews, as a designer or developer.

Interviews in this study have the following phases: (1) priming to the topic by describing the key concepts (mainly: UX, design…), the aim of the interview and case study and how the collected data will be used; (2) background questions about the participant; (3) main interview questions; and (4) end with a summary (see Runeson & Höst, 2008; Seaman, 1999). Semi-structured interviews contain closed-ended and open-ended questions (and follow-up questions) and in this study they are organized around themes about how people communicate and form shared understanding. Open-ended questions will elicit subjective and story-like, instead of binary, answers and the themes—the "way of doing"—will essentially form a relationship between the research questions and interview questions (Runeson & Höst, 2008). The questions are tailored to different backgrounds and roles in the project. The interviews lasted one hour each and they were recorded, transcribed and analyzed. The interviews were conducted face-to-face and took place at the business park where the company was located. Interviews were conducted in Finnish and in English and Finnish quotes used in this thesis were translated to English.

Data in this study is coded and analysed with a constructive grounded theory (GT) method (Charmaz, 2014). Constructive GT is a newer variant of GT and therefore

chosen as a methodology for this study. This paragraph will present the common characteristics in GT methodologies and the next paragraph will explain how constructive GT is used in this study. GT in a nutshell is an inductive paradigm to generate theory from data of any kind (Stol et al., 2016). It was initially developed by Glaser and Strauss (1967) and, according to Stol et al. (2016), now there are three main versions of grounded theory: Glaser's GT; Strauss and Corbin's GT and Charmaz's constructivist GT. GT case studies do not have a theoretical framework but they begin with the data (Olson & Kellogg, 2014, p. 30; Corbin & Strauss, 2008, as cited in Runeson & Höst, 2008). GT recommends open-mindedness towards the data by downplaying the role of initial literature review so that the data is not as likely used to test existing theory. However, Dey (1999) famously quoted that "'there is a difference between an empty head and an open mind'" (in Charmaz, 2014, p. 117), so clearly GT requires some initial literature review instead of none. GT acknowledges that the researcher's subjective beliefs and experiences will affect coding and the outcome of the research (Charmaz, 2014, p. 118; Stol et al., 2016). Validity and limitations of this study and GT as a method is evaluated in Sections 5.2 and 5.3.

Coding in constructive GT starts from initial coding—or open coding—which are then clustered into categories and ultimately integrated into cohesive theory to provide an answer to the research questions (Fox et al., 2008; Stol et al., 2016). To stay grounded to the data, constructive GT advice to coding for action in the initial coding phase. The codes in constructive GT should tell what the source has said—in this study the data is interviews—rather than classifying the data into existing categories (Charmaz, 2014, p. 116; Stol et al., 2016). Frequency of a code is a way to measure the explanatory power and how suitable it is for categorization: when a code is mentioned in many situations and by many sources, it is more generalisable than a code mentioned by a single source. Therefore the initial codes should be about the action because each source can have a different relationship towards the code, contributing crucially to the explanatory power. In practice, the sources might use the same word for different meanings or tell the same thing with different words, thus constructive GT instructs to use gerunds to preserve the context of the code. An example of such code in this study with explanatory power is

"reflecting the user experience in the system's quality attributes" (Charmaz, 2014, pp. 120–124; Olson & Kellogg, 2014, p. 32, 2014, p. 34; Seaman, 1999).

The case and participants will remain anonymous so that they cannot be traced from the finished thesis. The company and participants were asked to consent to record the interviews and use the data for scientific purposes which they all agreed. The interview voice records were removed after the study was completed. Only the researcher of this study had access to the data.

4 Results

The results are reported as follows: Section 4.1 will present the phase in the studied case where the system's UX was conceptualized. It will describe the problem, users and their needs that the participants identified and decided to converge to. Section 4.2 will outline which quality attributes of the system are important for particular users (for example, learnability is important for staff members). It will also evaluate study participants' own UX definitions about the product. Sections 4.3 and 4.4 build upon the contents of Sections 4.1 and 4.2 and together they will provide answers to the research questions (see Section 1.1). They will present the team's process and the utilized artifacts in the studied case as well as the phases within the process which are essential to achieve desired UX in the studied case.

4.1 Problem Conceptualization in Up-Front Design

The company was founded in Spring 2018 and it consisted of participants 2, 3 and 4 until the Spring 2019 (hereafter they are referred to as "founders"). This time period was the conception phase. The founders spent substantial time to understand the needs of their customers by iteratively validating, rejecting and reassessing their initially stated hypotheses. The hypotheses were not formally documented or measured but shared and refined among the founders. Participant 3 describes how they observed customers:

> *"So we had paper prototypes, with which we then walked to the rental shop. It was this small bike rental company in Helsinki and like… We had met them before and they always had, like, given us time to do a little more user testing with them. In practice, we simulated for them these, like, protos and collected comments, feedback. And at the same time, customers walked into that rental shop and we stepped aside and just watched and investigated when they were actually renting… We kept getting new scenarios and situations like, what we know and what we should solve. It was really valuable."* (Participant 3)

Towards the end of the conception phase the founders had accumulated enough knowledge about the users' needs. They had the necessary understanding what kind of user experience they are able to design for, that is, how to provide desired user experience meanwhile the customer fulfills her need by using the system. In other words, they had used this time to conceptualize the problem. The work was mainly non-technical to this point: the founders worked "in the wild" and focused more on producing lo-fi prototypes instead of a real system, although the early technical implementation started almost at the same time. The work towards increased understanding of the customers' needs also made some of the customers more familiar to the founders whom Participant 3 refers to as "lead users". Lead users in the studied case are proactive and will provide quality feedback—which is potentially transferable to the rest of the customers—and they are more willing to adopt early to new and unfinished features. The consequences of this approach are further elaborated in Section 5.1, here it is adequate to conclude that assembling this feedback loop up-front made it easier to incorporate it into the process the team started to follow in the post-conception phase (in Section 4.3) and that by collecting feedback early they were able to fine-tune the solution to be mature enough for larger crowds.

As stated at the beginning of this section, the company consisted of only the founders during the problem conceptualization phase. It is relevant to mention that new potential recruits were evaluated in the light of their orientation towards UX whether their role and occupation was developer or designer. When new members joined the team, they participated in an observation session with the customer:

> *"Get to talk and communicate with the right customers and users so that from there, like… So that the information doesn't flow too high from top to bottom for as many as possible. But that as many as possible have the ability to pick up that true understanding and information from that customer and share it."* (Participant 3)

4.2 System's Quality Attributes and UX Definitions

It was found that the team in the studied case reflects continuously, yet partially unconsciously, the quintessential user experience in the system's quality attributes (Table 4.1). Quality attributes in the studied case are the implicit glue that binds the feature-level user interaction with the technical "under the hood" work. When asked about what makes their system easy to use, frequent answers hinted about the existence or awareness of a quality attribute, such as learnability. Here is a practical example how the team talks about learnability: Based on the fieldwork in the conception phase, the founders detected that their customers hire seasonal workers (staff members), who are one user group of their—or competitors'—product, so learnability has a twofold importance: (1) the team is able to design better UX; and (2) differentiate themselves in sales negotiations from the competitors whose products take longer to learn (one of the company's goal is growth).

Table 4.1: Quality attributes the participants used to describe the product

User	Quality attributes
Consumer	Efficiency Usability
Staff member	Learnability Efficiency Usability
Shop owner	Administrability Maintainability Reliability

Table 4.1 reveals an aspect of the product which became apparent during the interviews: the product has instrumental value for the users and the team treat it as such during design and development (see design for experience in Section 2.1). Participant 3 explains that the consumer's user experience is good if the total time for checking in and collecting the equipment is kept reasonably short (=efficiency), so, at the time of the interviews, for example increasing engagement or consumer's time spent with the product would be intrusive and not provide additional value for the consumer.

Participant 4 envisioned how maintainability and reliability—product's internal quality attributes—can contribute to the user experience and affect design decisions even though it does not bring immediate benefit to customers: if they hire new employees and their customer base will grow fast, thinking beyond the current scope and ensuring maintainability and reliability can have positive impact in terms of release speed, number of defects and accumulated technical debt. Table 4.2 and 4.3 gives a more detailed look onto participants' descriptions of the product's user experience and of what problem the product solves for the users. Comparison between the quality attributes (Table 4.1) and descriptions (Table 4.2 and 4.3) will in part show the state of shared understanding of the product's user experience among participants.

Table 4.2: Role of UX, paraphrased general and product-specific UX definitions

Participant	UX in participant's work	Own general UX definition	UX definition for the product
1	Important but not primary	Solving a problem quickly and efficiently	User understands that she interacts with the brand
2	Providing feedback	Intuitive and delightful	Easy-to-use (cognition)
3	Central	UX can't be created, it emerges by designing for experience	Learnability; Enabling staff members to be present with the consumers
4	Managing resources and focus for other people	Fluency; Recognition of what tasks user is able to do now (see "affordance", "signifier")	Embedding understanding of the fluency into artifacts
5	-	-	-
6	Central	Everything the user sees and thinks when she interacts with a system	Understanding the user from the feedback
7	-	-	-

Participants' general UX definitions and UX definitions about the product varies greatly (Table 4.2). Their roles are different and the company does not have an official UX definition so variation is expected. However, the participants generally described the company design-oriented and unanimously listed UX as, not the only but nevertheless, an important goal. UX definitions for the product also vary in how some participants focus on the product and the artifacts and some participants talk about the *experience* of the user (see Section 2.1 for a discussion between user experience and experience). Table 4.3 shows some agreement across participants of what problem the system solves. It is notable that many of the described problems are more about the customer's

business aspirations instead of the consumer's needs suggesting the UX as an important goal is not as clearly articulated and shared among the team.

Table 4.3: What problem does the system solve?

Problem	Participants
Enabling add-on sale	6
Helping in inventory management	1, 2, 4, 6
Increasing consumer knowledge from the data	1, 2, 3, 4
Renting in advance	4, 6
Replacing multiple existing tools and paper	2, 3
Speeding up the rental process	2, 3

4.3 UX Confluences Between Developers and Designers

Below is a simplified representation of the iterative software development process used in the studied case which the whole team follows (Figure 4.1). By analysing the interviews, three distinct phases within the process were identified (Figure 4.2) which are relevant for RQ1: (1) Design Intent; (2) Design Handoff; and (3) Quality Assurance. The figures (4.1) and (4.2) were deduced by analysing the interviews. This section will decipher what happens in the process during these phases as well as what artifacts are used in collaboration between developers and designers during them. Some key artifacts are introduced here but a more in-depth analysis is in Section 4.4.

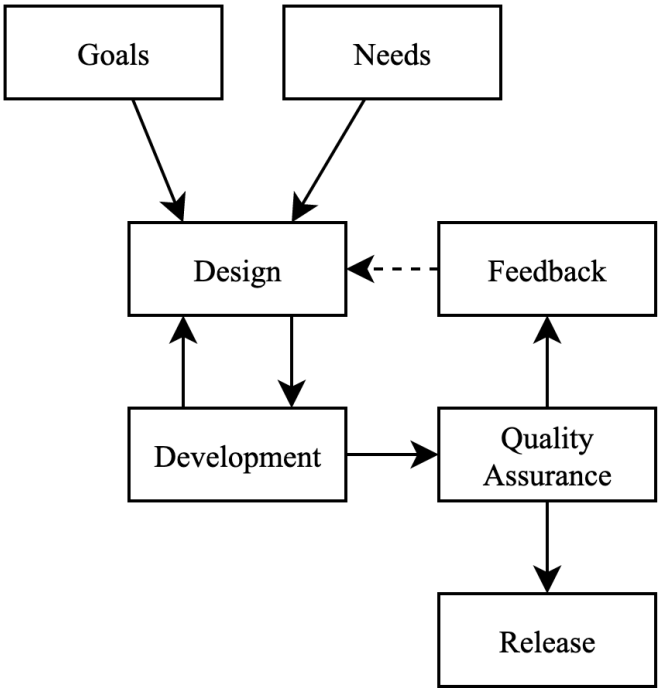

Figure 4.1: The team's iterative software development process

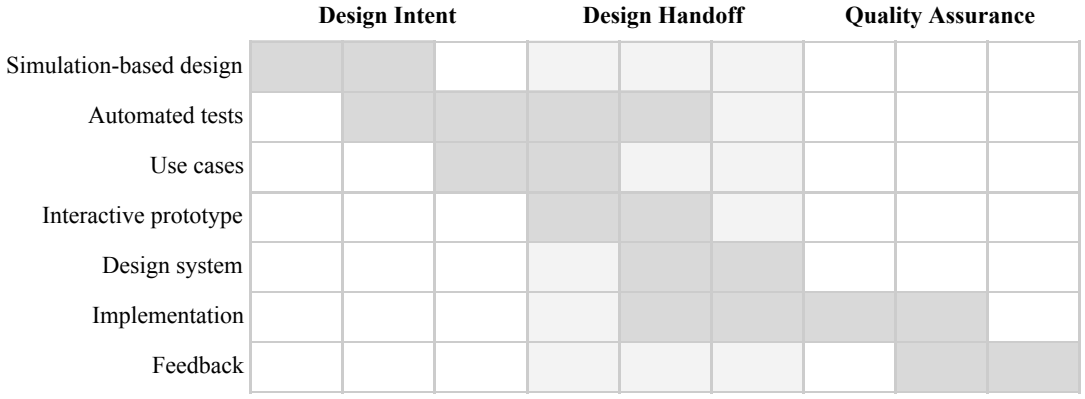

Figure 4.2: UX Confluences Between Developers and Designers

It was found that in the studied case developers and designers form shared understanding of the system UX under development by UX confluences depicted in Figure 4.2. They are the activities within the process (Figure 4.1), where developers and designers collaborate in order to achieve desired UX. Next, the confluences are briefly presented, followed by rationale later in Section 5.1 why they are important for the shared understanding of the UX in the studied case.

"Design intent" is the first confluence and it happens at the start of the team's iterative cycle. The company in the studied case is design-oriented and the team involves customers in the process as much as possible. In this confluence, the team tries to reflect customers' needs to the system's quality attributes and the designers do this by simulating customers' behaviour with lo-fi paper prototypes. In this confluence, the team tries to include as many diverse opinions as possible to think about the problem and then to converge into a single solution. Some of this work is documented into artifacts but some of it is, what Participant 4 calls, "learned implicitly by working together". This confluence emphasizes the social dimension of software engineering and team interdependence, as described by Participant 3:

> *"When we have gone through these joint sessions, demoed to each other, like 'how this works', then everyone has been able to bring missing situations into it, like 'what if this situation happens here, what should I do here?' Then we have completed them into the proto. In other words, it has been possible to discuss, 'is this a realistic point of view for the customer, would they need to solve this?' And it has also been really important to us, that we have had that good customer understanding that we haven been able to take, let's say, ten customer scenarios and say 'these two are the truly important scenarios, let's do the two and forget the rest.' They don't actually happen. And because of that we've been able to 'ship' quickly new releases for the customer to test and to learn more."* (Participant 3)

This was in part possible because the founders had established a relationship to customers they refer as lead users:

> *"Then we just called the customer."* (Participant 3)

> *"The customers might have had a little request, like 'we need a feature like this for this', but that wasn't enough yet, we kind of wanted to dig deeper to what the problem behind that wish was. And they had the time and patience to go through those conversations with us. So we were able to think again on the basis that, well: the way they do now is not necessarily the smartest way to solve it. So we might have come up with something entirely different, allowing them to solve the same need and problem later. But it was valuable precisely because we had a feedback loop running and the customers had the time and patience to also talk to us."* (Participant 3)

Participant 4 describes the work in this confluence:

> *"So we always go into the root causes in the discussion—we do not take the shallow 'I need this', but 'why you need this'."* (Participant 4)

Participants 3 and 6 say that while they do not implement everything, their implementation supports most things through automated tests which ensure core functionality works amid changes (see maintainability and reliability in Table 4.1). Participants 3 and 6 respectively continue:

> *"Our product is built on top of the… so that it will continue to work when changes come … Which also means that the problems we solve are, like the most challenging: so when we successfully solve the customer problem, those features and the product development will also benefit the smaller customers and their needs."* (Participant 3)

> *"Test-driven development has been, like, our big thing. So that based on the tests, we know what should fundamentally work in the most important cases."* (Participant 6)

However, Participant 1 expressed concerns about the mismatch between the design intent and the actual implementation:

> *"Then you solve it maybe a bit later and that's always risk: leave things later. When are you gonna do that then?"* (Participant 1)

Nevertheless, Participant 4 said that it is pretty easy to "feel" the priorities because the survival of the company depends on making the right calls in terms of what value they are able to create for the customer:

> *"Being in a startup is easy in the sense that you have colder realities: you have less runway, you have as if less of, 'we could postpone this for a year and a half' attitude. That is, it will often be that you are forced to generate more resources for yourself that are either cash flow or reaching milestones, which will allow you to invest or else you will die. And there is no need to discuss further prioritization at that point. Then like… This usually actually makes things pretty much easier. You're like, 'these are now a bit like must-have stuff from these perspectives'. And yes, we just have to put up with (shit?) or pain. Then if there is kind of like prioritizing between them, then I think it often comes from— there are, of course, all the MoSCoW methods and others that can be used to get it or play some poker, but often it's about how we obtain… It rarely goes wrong, when you have equally prioritized stuff, then which of these will produce the biggest impact and value for the customer the fastest, because they will always reflect customer value after all. And if I can break it down into smaller pieces, I can divide it into this and get a less-priority piece of it and get a high-priority piece. And and… That's how I think it is…"* (Participant 4)

"Design handoff" is the second confluence and at this point, the team has written use cases into the backlog for this cycle and concluded that the design is "finished". Design is represented as an interactive hi-fi prototype in Zeplin and InVision tools. The distinction between artifacts in the previous confluence and this is that now they are, according to Participant 4, "boundary objects": elaborate and abstract artifacts that do not necessarily help to fix a problem if other team members do not understand them, but may in fact increase miscommunication and difficulties in coordination. Participant 4 talks how the team uses boundary objects to manage complexity, detect patterns and build shared understanding, acknowledging that "outsiders" are very likely unable to derive any meaning or purpose from them. In other words, the team converges into a single solution and tries to explicitly specify the intention in the artifacts.

The rationale to call these distinct phases "confluences" is apparent with design handoff: it is a continuum, not a point in time, as the name might suggest (see Figure 4.2). The people, developers and designers, collaborate and produce artifacts during the activities in confluences. In the studied case, the designer will educate the developer about how a representative user (see "persona") will interact with the system whereas the developer will use her expertise and ask many "what if…?" questions about the general flow to fulfill user's needs. In comparison, Participant 1 described the worst kind of collaboration between developers and designers:

> *"The worst situation is if designers sit in their design team silo. Do their designs and push a button and have it in Zeplin or anywhere and that's it. And then developers will need to basically just take a look at the designs and start to create their understanding on what has been created and then to start their implementation. There's no communication in between."* (Participant 1)

"Quality Assurance" is the last confluence and it is at the end of the cycle. Participant 6 points out how even the high-fidelity prototypes that look and feel as if they were real are only abstractions and that the final, "product-level fidelity" is achieved when the implementation is almost finished: Participant 6 explains how the prototype is a reference for him towards the final solution and, as the implementation approaches the design intent, previously unknown behavior will emerge. Participant 2 adds that even the high-fidelity prototypes will inevitably miss some aspects; it is difficult to cover all the interactions, such as errors. Instead of releasing immediately, the developers will do quality assurance first by peer reviewing the implementation and then by demoing it both internally within the team and externally to some customers to receive feedback. Depending on the amount of new knowledge the team will gain from the implementation, they can decide to do another design cycle if it is necessary. While internal quality assurance is part of the team's process, the acceptance criteria in the studied case is mainly based on the developer's own values and interests, that is, some are more geared towards UX and design than others (Table 4.2) and will review them with more discipline. However, when asked *"How relevant is UX in your current work?"*, Participant 6 stated very clear relationship between user experience and developer:

> *"Well I think it is very relevant for a developer's job: because it's like the final outcome, that I have made which the user then will use and experience."* (Participant 6)

Participant 4 expressed in an answer to the same question why diverse opinions and bi-directional communication about the core idea at this final stage are needed in order to preserve the holistic perspective and user's conceptual model:

> *"Because sometimes I make surprise attacks, like: I'm the one farthest from our product in our entire organization right now, so it's easy, it's easy for me to sit in a situation or review and start participating in one because I'm closest to the neutral user. And the fact that if I have been '2 days' away from the product and that—some basic stuff that might just scream in my face, doesn't seem understandable, because I have not thought about that problem for two days. So that's again perhaps my [role] towards usability."* (Participant 4)

Participant 6 also recognizes same:

> *"It's also a constant kind of reminding myself that the more I do it [as a developer], the more blind I get to it, to those problems, because I'm starting to remember what my challenge is. Maybe it's like what comes to my mind, that developers have a big responsibility for it [UX], but at the same time you have to always break like your own bubble from that UX"* (Participant 6)

Finally, the team invites as many customers as possible to a demo session where they are able to receive qualitative feedback—and new perspectives. The customers are able to express any opinions they have which the team uses as a validation on their hypotheses: they have encountered situations where a customer said that the newest increment will in fact impede them to fulfill their need putting the team "back to the drawing board". This attitude towards increments as a mechanism for validated learning is elaborated in Section 5.1.

4.4 Artifacts Utilized in the Collaboration Between Developers and Designers

This section will review the relevant artifacts, at the time of the interviews, for the collaboration between developers and designers in the studied case (Table 4.4). The perspective towards artifacts in this section is akin to how repeaters receive and retransmit a signal in telecommunication: signal here is the information about the user's need, the design intent, and it flows through the artifacts (repeaters) ending up into the deployed software. Like signals, the design intent is prone to entropy as it flows through people and time so the purpose of the artifacts is to represent the current state of the design intent.

Table 4.4: Artifacts utilized in the collaboration between developers and designers

Tool	Who makes it?	Who uses it?	What is it used for?	When is it used? (confluences in Figure 4.2)	Notation	Fidelity
Paper	Designers	Designers	Simulation: Use cases; User flow	1	Informal	Low
InVision (design tool)	Designers	Developers	Interactive prototype	2	Formal	High
Zeplin (spec tool)	Designers	Designers; Developers	Handoff; UI design; Asset inspection	2	Formal	High
Design system	Designers	Developers	New features have essential design language; Generalised components	1, 2	Formal	High
Backlog	Designers; Developers	Designers; Developers	Link design with use cases; Prioritization; User stories	1, 2	Informal; Formal	Low; High
Physical board	Designers; Developers	Designers; Developers	Prioritization	1, 2	Informal; Formal	Low; High
The product (implementation)	Developers	Designers; Developers	Onboarding; Feedback; Evaluation; Quality assurance	3	Formal	High

As mentioned in Section 4.3, the team follows a single process (Figure 4.1), which, as the Participant 1 tells, will keep design "on par with development". Considering these circumstances and the dissimilar nature of development and design workflow, artifacts can distil people's subjective beliefs into shared understanding (see boundary object), or, as Participant 4 describes:

> *"It involves a lot—where we are probably going here—artifacts and other things, and things related to basic psychology, that you kind of have this shared (paper-ops?) which is bought and which gathers people around to discuss instead of my opinion against yours like this."* (Participant 4)

When new people joined the team, they used artifacts to form a conceptual model of the system instead of a formal onboarding. Depending on the role, people used different

artifacts but usually it involved testing the product in particular. As described in Section 4.3, the "product-level fidelity" seems to represent the most accurate design intent.

The design workflow starts with a task analysis about the user's mental model with simulation-based user interface design: the designers use simulation to ensure the user flow is cohesive and to transform the holistic nature of the feature from the user's perspective to the developer. It is lo-fi and aims to yield an abstraction which would solve many scenarios from multiple users. Sometimes the implementation directly follows the simulation without hi-fi prototyping:

> *"I think it's important that they're [developers] involved in that kind of design and you can simulate it for the developer and user if you're user testing it so that it really does all the things you imagined and really wanted it to happen when you designed it… So, somehow it stays holistic and you simulate it 'end-to-end' with all its intermediate stages, so you don't stay on assumptions at the point when someone starts developing it, that… 'That's how it should work, this should happen'."* (Participant 3)

It appears to be a personal preference, though, to either use lo-fi or hi-fi prototyping at the start but the participants, who are in favor of lo-fi prototypes, advocate the rapidness and, more importantly, how lo-fi prototypes will not derail the focus and the tone of discussion from the most important topics during this early phase of work:

> *"But that—let's say—basic principle is that we have demoed and simulated, and gathered feedback from each other… that principle to design has remained. What feels a little bit like, as towards the summer [of 2019], has shifted more to those like the utilizations of digital tools at an early stage, which perhaps for me feels at times that it may have been perhaps a little harmful. You feel like making changes and fixing them is slower and, like, the focus in those conversations easily goes into a little bit of unrelated things. Not necessarily for things that are so important at the moment, so you, like… We've been cautious to stay in that really crude world of paper prototyping, where you can take a new paper or a post-it note right away and immediately draw a different solution and just draw big X over some things and fix things together in it… Like the "loop" and the iteration is much faster and the focus really stays like… really in how the core functionality and logic should work from the point of view of different users and not so much stressing over small visual things, which is also important at some point, but not necessarily at that very moment."* (Participant 3)

Next, the design team aims to create a "production-ready" interactive hi-fi prototype with a combination of Zeplin and InVision.

Eventually the design is linked to a written issue in a backlog. Here, the holistic design intent is split into multiple issues and then prioritized.

The type of artifacts utilized are subject to change as new needs arise. It may not be exactly clear, how the artifacts are used or which artifacts are not as useful, however. At the time of the interviews, Participant 2 was unsure of the purpose of InVision when they were transitioning to use Zeplin more whereas Participant 6 listed InVision as the most useful tool to investigate the feasibility of the design.

5 Discussion

This chapter summarizes the findings presented in Section 4 and more specifically what makes the confluences presented in Section 4.3 important for UX. During "*Design intent"* a lot of prioritizing happens. The team narrows down the scope and converges into a solution which is represented as an interactive prototype. This means that some of the essential knowledge about the UX is not documented anywhere and just gets lost. On the other hand, the developers should aim to represent the design intent with their implementation, not just the prototype and use cases they receive in the *"Design handoff"*. This is an important finding in this study. The developers are to a large extent responsible for the quality assurance. And, obviously, it is the implementation, not the design intent the customers will use. And when the developers are part of *"Design intent"* confluence, they are able to draw a distinction between the core and peripheral functions of the solution so they can start designing architecture and writing automated tests for the core functionality instead of being unaware of the customers' needs before they will provide feedback for the finished implementation. Prioritizing and converging in the studied case means that they will implement some of the ideas now and others later which means that there will likely be changes. By testing the core functionality, the developers are more prepared for changes and can also avoid regression and the need for longer re-iteration but instead "build the thing right" at the first time.

Returning to the analogy about the user's need as a signal and the artifacts as repeaters (see Section 4.4). The work starts with informal lo-fi artifacts ending to the highest fidelity and most formal representation of the design intent in the form of a finished product. The participants have described how the "notation" and "formality" of discussion will—or should—correlate with the artifact in use so that they will focus and solve the right problems. The finished product is no different from the other artifacts in a sense that while the intermediate artifacts are a means to an end to implement the product, the finished product itself has instrumental value to the user experience—it is not the experience, it is designed for experience.

5.1 Related Work

On the system level, external simplicity increases internal complexity. Good-looking user interface (UI) is essential, but not enough, “the design must be thought of as a total experience” (J. Patton, 2002). More features do not cause the UX to improve and retrofitting UX is not feasible since UX is not a feature, but will likely cause difficulties. Quality is not more, what matters is the outcome. Feature creep makes the system more complex and the original design intent is lost, the system is “featured to death” making the UI among the most complex parts of the system (Heath & Heath, 2007, pp. 48–49). (Norman, 1998, p. 171, 1998, p. 172, 1998, p. 196, 1998, pp. 204–205, 2013, p. 293; J. Patton, 2014, p. xliv.)

During *“Design intent”* confluence, there are many unknowns, the team has many questions to ask and answer: “in real-world practice, problems do not present themselves (to the practitioner) as givens … [A practitioner] must make sense of an uncertain situation that initially makes no sense” (Schön, 1984, p. 40). Many questions which the team can conveniently come up with an educated guess by relying on its own intuition and use as a baseline for solutions. Regardless of the success of this approach, the same underlying need may manifest in multiple ways, therefore User-Centered Design guides towards considering the user’s feedback as a starting point for problem setting or framing. Thus, an essential component of this confluence is to question the feedback: decoupling the manifestation from the primitive need by not taking it for granted in order to identify the root cause and to avoid implementing potentially unnecessary features. Here, the quality attributes have a significant role: when the team is able to reduce the problem into its fundamental particles, they can improve the overall user experience in regards to the particular quality attribute and ultimately resolve the problem by reflecting the expressed problem to the quality attributes. A repercussion of user’s feedback taken at face value is that it can potentially result in contradicting quality attributes.

Startups are “fighting for survival”, and influenced by competitors so it is important for a startup to try to attract users from competitors and being aware of potential users'

preconceived needs (Hokkanen et al., 2015). In the case studied in this thesis, influence of competitors occurred when the participants talked about what kind of systems the potential users have accustomed to use. Users are already familiar with customizable and versatile solutions but the participants realized at the same time that the desire for customizability would come at the expense of efficiency and learnability. Therefore, taking the feedback at face value could weaken the system cohesion and core functionality. As mentioned in Section 2.1, proposed design solution is not necessarily best even though it would satisfy all the requirements because many ideas could achieve the same functionality and, in theory, one of them is a "perfect fit". If a system under development is missing features, it is a confirmation of the correct hypothesis, not a sign of failure. If the most important thing is implemented properly, other areas can be "good enough". (Norman, 2013, p. 238; J. Patton, 2002; Ries, 2011, pp. 65–66.)

"Design handoff" confluence is principal for user experience in the case studied in this thesis because the design intent is irreversibly split into its constituent parts (into use cases) which will pose a threat in a form of degradation of the intended UX. This polarity of holistic user experience and atomistic realm of machines—a paradigm shift from usability to functionality—has been noted in user experience literature (e.g. McCarthy & Wright, 2004, as cited in Benyon, 2013, p. 95; Cooper et al., 2014, p. xx; McCarthy & Wright, 2004, as cited in Hassenzahl & Carroll, 2010, p. 4; Brhel et al., 2015, as cited in Larusdottir et al., 2017, 2017; J. Patton, 2014, p. xiv). This split bears resemblance to the inevitable information loss in data transformation, such as in quantification of qualitative data (Seaman, 1999). Larusdottir et al. (2017) write that the user stories do contain some information about the user's interaction with the system but more formal use cases lose this context. They continue that user stories are a misnomer at times and how they "are rarely applied as a tool for interacting or collaborating with users. On the contrary it may again be used as a substitute to proper UCSD [user-centered systems design]". Salah et al. (2014) add that the difficulty is in fact the split, not absence of usability as an important goal. Splitting the work is necessary in the case studied in this thesis, however, because the team is committed to release and receive feedback faster. Fundamentally, in the case studied in this thesis, the artifacts should transmit sufficient knowledge for the developer to be able to write automated tests for the core functionality and to ensure quality assurance. If developers

are part of the Design intent confluence and if quality assurance is incorporated into a project, the developers do not essentially need to implement proof-of-concepts or one-off solutions but can focus on building cohesive production-ready functionality.

Not every developer is design-oriented or interested in design or has time for design work, however, and they are generally only part of the *"Design handoff"*, which is also acceptable but not necessarily good for the shared understanding of the UX (Beyer et al., 2004, as cited in Magües et al., 2016, 2016). As was in the case studied in this thesis, some developers actually hope for complete design so they do not have to do any of it themselves—or participate in horizontal collaboration—but it adds more demands for the quality of artifacts in design handoff. While some artifacts are proposed for this purpose, it remains problematic (Magües et al., 2016).

"Quality Assurance" confluence. Larusdottir et al. (2017) report in their review that responding to change due to usability evaluation will become progressively more difficult the closer the solution is to being functionally finished. The result from the review indicates that a project would benefit from "Definition of Done" and from a designated role for quality assurance in terms of usability if usability is an important goal. Moreover, how and when to involve the user or customer in the process varies. In UCD and in the case studied in this thesis, users are part of the start of an iteration and evaluation with intermediate prototypes. In UCD, intermediate artifacts provide a rudimentary interface for people in their most natural context whereas Agile disregards documentation and invites them to evaluate the functional increment. As stated in Section 2.4, it is better to frame and solve the problems with inexpensive and malleable prototypes than with the actual software system. Generally, and in the case studied in this thesis, there is little formal usability evaluation. (Agile Alliance, 2001; Dhandapani, 2015; Larusdottir et al., 2017; Magües et al., 2016.)

The importance of canonical use of abstractions and artifacts during problem solving became evident in the case studied in this thesis. The team is able to judge, what are the essential functionalities they should implement immediately and what can they leave for later. They aim for a flexible and extensible solution and architectural decisions so that they can keep up with the demand from the customers and preserve code quality and cohesion. However, this focus on more immediate benefits may be detrimental to UX in

the context of iterative development: when things are deprioritized and postponed, their *raison d'être* is gradually forgotten from the collective memory, never receiving the necessary attention. From a designer perspective the challenge may manifest itself in the holistic nature of the UX: the design intent should be stored in the artifacts without excessive information loss, such as into an iterative prototype and use cases into the backlog as in the case studied in this thesis, since it is not in people's minds anymore. From a developer perspective the challenge is to avoid implementing peripheral or arbitrary features instead if they are unaware of the actual users and their needs. (Larusdottir et al., 2017; Magües et al., 2016.) From the perspective of the software system under development the shared understanding of the UX is weakened. The things that are *not* done are perhaps more important than they seem.

***

There are generally two distinct ways for collaboration between developers and designers: single process or separate process. Sometimes the process is selected organically, other times it is externally imposed by the corporate culture, leaving the practitioners little control over the success of the team and the product. How the rest of the organization and stakeholders behave and what processes are followed elsewhere may also inhibit or aid the work of a software development team. (Ferreira et al., 2012; Kuusinen & Väänänen-Vainio-Mattila, 2012, as cited in Larusdottir et al., 2017.) The case studied in this thesis represents a single process and in the separate process there usually is a clear design handoff event where the parties briefly unite to discuss the design specification.

Ferreira et al. (2012, 2010) found in their case studies that unified development-design process increased shared understanding of the UX vision and challenges and that the work towards the UX vision included and benefitet from bi-directional communication and feedback both from development and design perspective. Similar findings were in a literature review about Agile and UCD integration (Dhandapani, 2015). As in the case studied in this thesis, developers are not necessarily part of every UX discussion but they have expertise about the possibilities and capabilities of technology, not only the constraints and limitations (J. Patton, 2014, p. xii). This is sometimes a bit overlooked

aspect of the nature of the work and the developer's role can be somewhat demeaning: other people will enforce the implicit status hierarchies and dictate the decisions leaving developers as mere subservients, silently implementing anything they are told to in time-pressure (see Section 2.2). Although it is quite evident that the end user does not need to know about the internals of the system, the role of the development process should not be overlooked because that is what the user will eventually interact with.

Processes are typically aligned by time dimension which goes only to one direction. Design thinking is as much of defining the problem as coming up with a solution, so the process moves back and forth, not in "micro-waterfall" linearity, but in waves of divergence and convergence. Ferreira et al. (2012; 2011, as cited in Larusdottir et al., 2017.) report that the collaboration between developers and designers is "ongoing", that is, developers and designers switch between the tasks of the two disciplines according to their competence: they work on their own assignments and support and assist on other people's work on demand. Similar results were found from the case studied in this thesis, called UX Confluences.

Proponents of separate processes (see Ferreira et al., 2010) express that the designers are free from the oppressive logical confines of software development work without hampering their creativity. Designers also avoid adjusting the holistic and sporadic UX work for short iterations. Considering the common UCD techniques, on one hand the UX professionals can't incorporate them into the tight schedules, but on the other hand merging them may be inappropriate: the outcome of UX work does not always directly contribute to the software system under development or the UX professionals are not full-time members of a software product development team. (Beyer et al., 2004; Ferreira et al., 2010, 2012; Larusdottir et al., 2017; Magües et al., 2016.) Anyhow, the developers have to deduce the logic of the design (with feedback from the designers) once it is finished and handed off to them (Ferreira et al., 2010). Considering the tight schedule of development work, it is perhaps no surprise that also developers often find themselves in a situation where they do not have time for UX work (Magües et al., 2016).

Although plenty of literature about UCD techniques in Agile exists (see Magües et al., 2016), there is no consensus on what, when, how and for how long UCD activities and

user participation are preferably performed if they are combined with Agile development's short iterations (Magües et al., 2016; Salah et al., 2014). Beyer et al. (2004) advocate for an altogether separate (time-boxed) up-front divergent design phase, such as contextual inquiry, followed by a more traditional and convergent iterative software development phase as in the case studied in this thesis. The opposition for "Big Design Up Front" comes in part from the software system's emergent properties: the requirements are elicited along the development lifecycle (Magües et al., 2016; see also Section 2). In the context of software development lifecycle however, people's needs are somewhat static: their existence does not depend on the presence or absence of a software system (Beyer et al., 2004). Therefore they will likely remain largely unchanged throughout the system's lifecycle and are convertible to the system's overarching quality attributes as in the case studied in this thesis. Even though usability would not be necessary, "it is always cheaper to do the job right the first time" (Crosby, 1983).

***

In an abstract business perspective, the existence of an early-stage startup is determined by their ability to convert people into paying customers and a definition for a successful product is "one that encompasses any source of value for the people who become customers" (Ries, 2011, p. 28). Startups do not usually have excessive resources initially so they need to be wary of not producing waste, for example, unused features or non-value adding artifacts. Furthermore, startups might operate in a new field of business or work with new product development with some degree of uncertainty where they are continuously finding the problem fit and learning more about who the potential customer is and what the potential customer wants (Hokkanen et al., 2015). Achieving high quality is expensive and time-consuming (both scarce resources for a startup) and matching it to the potential customers' needs is not entirely obvious. Hokkanen et al. report (2015) that the startups should have a UX strategy so that the quality—at least on the UI surface level—of the initial launch does not interfere with conveying the product idea to the potential customers. However the early adopters are more willing than the

majority to participate in the design and tolerate poor usability as long as the system fulfills their needs, as were the “lead users” in the case studied in this thesis.

Eric Ries has also written about aforementioned realities of a startup in his book, *The Lean Startup* (2011) and the case studied in this thesis was also in this kind of situation at the time of the interviews. In the context of startups—and more broadly in new product development and in UCD—the overarching goal for a sustainable business is a minimum viable product (MVP) and journey towards it is to consider the design and development until the first stable version as validated learning about the potential customers (Figure 5.1). When the team moved from phase 1 to phase 2 in the case studied in this thesis, software systems in general can publish a release candidate for larger distribution at that time (Larusdottir et al., 2017).

Hokkanen et al. (2015) write that up-front design that “aims at complete product design is not suited to the needs of startups” (i.e. phase 1). For example: pivoting to altogether different solutions or users would lead to wasted effort, missed opportunities and financial troubles. Additionally, many startups do not have the resources, that is, budget and skills as was the situation in the startups they analyzed. However, the case studied in this thesis had experienced people in UX during the initial phase and enough seed money to be able to conceptualize the problem with the potential customers and lead users. Ries (2011, p. 37) summarized the problem in clearer business terms: “What if we found ourselves building something that nobody wanted? In that case what did it matter if we did it on time and on budget?”

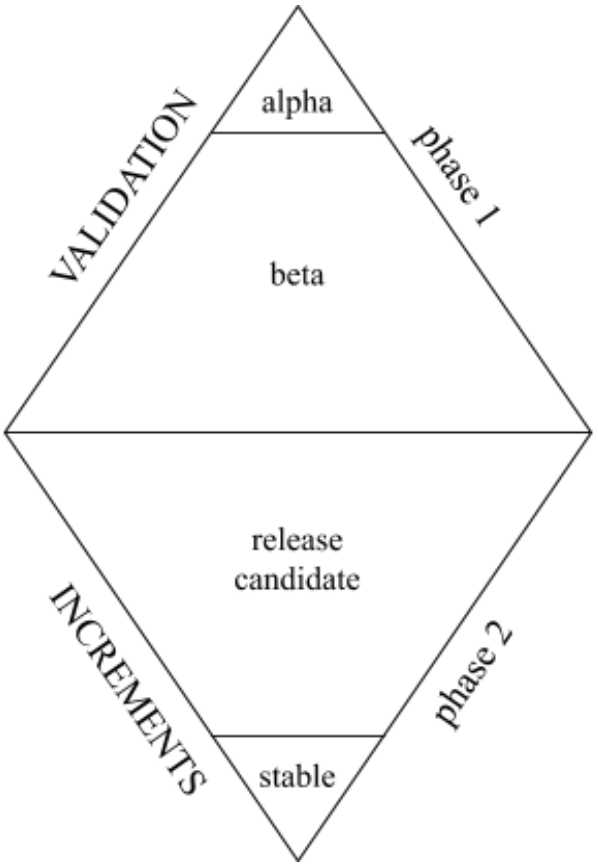

Figure 5.1: The conception phase (1) and the post-conception phase (2) in the case studied in this thesis synthesized into a representation of the whole system's lifecycle

5.2 Validity

Shadish et al. (2001) define the term validity as "approximate truth of an inference" (p. 34), that is, how close to the truth the results are. Creswell & Miller (2000) define validity in qualitative research "as how accurately the account represents participants' realities of the social phenomena and is credible". The authors add that validity refers to inferences about the data, not the [raw] data itself. (Creswell & Miller, 2000.) Shadish et al. (2001) explain that "no method guarantees the validity of an inference" but "method may affect more than one type of validity simultaneously" (p. 34).

Shadish et al. (2001, p. 38) describe four categories of validity: (1) construct validity; (2) external validity; (3) internal validity; and (4) statistical conclusion validity (which is omitted here). Construct validity (1) is about how the concepts are used in a study and how they are measured. External validity (2) refers to generalisability, for example: how representative is the study for generalisation or how transferable is the study, that is,

does it apply in other cases. Internal validity (3) considers the causal inferences. Shadish et al. (2001, pp. 9–12) make a distinction between causal description and causal explanation: a research should both describe what is the causal relationship and explain what are the conditions for causal relationship.

Creswell & Miller (2000) explain that—during the time of writing—there are no agreed guidelines on validity procedures in qualitative studies and present their own framework on how to choose them. The framework consists of two parts: (1) the lens used by the researcher; and (2) researchers' paradigm assumptions. The lens (1) means what viewpoint is used to achieve validity, and more specifically, "the views of people who conduct, participate in, or read and review a study" (Creswell & Miller, 2000). In general, the researcher, or a reviewer has to evaluate that the original researcher has collected enough data and whether the participants' perceived reality is accurately represented in the results. The paradigm assumptions (2), or world views, contribute to the status quo of what validity means. They have identified nine validity procedures suitable for the framework—most common are member checking, triangulation, thick description, peer reviews, and external audits (or audit trail). (Creswell & Miller, 2000.)

As with any other research, it is important to maintain the chain of evidence (or audit trail): the results have to be based on the data, not on preconceived notions of the researcher. In this study, the participants were offered to review preliminary results to avoid misunderstanding (see member check) which is, according to Lincoln & Guba (1985), "'the most crucial technique for establishing credibility'" (as cited in Creswell & Miller, 2000; Runeson & Höst, 2008). However, the proposed meeting to present the preliminary results to the participants did not materialize.

5.3 Limitations

Considering the four categories of validity in Section 5.2, the most pressing question remains "what is shared understanding?" (see construct validity). Since it is perhaps not the most suitable concept to define and measure objectively, an effort is made to provide enough context specific to the studied case. Similar limitations affect the term "user experience" as well. As the research progressed, I accumulated new knowledge

about the topic, starting from a rudimentary interest towards it. The learning and evolved attitude towards the topic is evident in the causality (see internal validity) where the results were readily explainable with the studied literature, no matter if it explained a true causality or just was fresh in memory and seemed plausible. This is not how Grounded Theory should be conducted which recommends not to test existing theory with the data (see Section 3.2).

Qualitative research has limitations (for misunderstandings about case study research, see Flyvbjerg, 2016). I base the following claim on my computer science studies and discussions with peers about qualitative research in technology: the results of qualitative research are not generalisable because the results are not quantifiable (see external validity). Moreover, comparing the qualitative results to related work either (1) “do not prove anything new” if the results are similar to the related work, or (2) “are not valid” if the results will prove something new. I do think, however, that qualitative research was a suitable approach for this thesis.

In order to land with more reliable results, a variety of data collection methods should be used. This is called triangulation and it reduces the possibility of drawing too hasty conclusions. For example, people’s answers to interview questions and perception of reality might diverge from data in textual form.

The initial plan for triangulation was as a matter of fact to interview and analyze artifacts but the interviews already gathered a lot of data about the artifacts. There can be a difference, though, in what people say and what they do, and the study may not be as valid by relying only on spoken word. Furthermore, the interview questions were open, that is, the answers were probably not as precise as what would be found out with additional methods. On the other hand, seven out of ten employees of the company were interviewed and they gave answers from different perspectives—not just “the happy path”. Additionally, as Seaman (1999) has stated: “Although a great deal of information can be gathered through observation, the parts of the software development process that can actually be observed are limited. Much of software development work takes place inside a person's head”.

6 Conclusions

This is an exploratory case study about the shared understanding of the system user experience under development and collaboration between developers and designers. The study used interviews as a data collection method to answer the two research questions: (RQ1) How do developers and designers form a shared understanding of the software system UX under development; and (RQ2) What are the artifacts utilized in their collaboration. This topic was chosen because collaboration between developers and designers is invaluable to ensure the success of a user-centered software development project and because the roles and boundaries blur faster than the methods and artifacts. Here are the central implications of the results.

ELICITING DESIGN INTENT. In a software development project nowadays, there is value in activities that do not directly contribute to the "next increment". The activities can follow the same process as the rest of the software development and most certainly share the same goals as the software development project and team. New product development can start with problem conceptualization activity where users and their needs are identified and the system's UX is conceptualized. It starts from the perception of designing for experience and that the system's role is minimal to the user experience at the beginning. These principles allow usability to be embedded as an intrinsic value of a complex functional system and the results of the study support the notion that an up-front design activity will contribute positively to the shared understanding of the system UX under development.

REPRESENTING DESIGN INTENT. Usability is, among system's other quality attributes, imperative for particular users. The quality attributes are reflected in the relevant artifacts, transmitting the design intent to the implementation. Also: communication creates and transmits information about the shared understanding. Depending on the success of the information transmission, conceptual integrity may degrade when the design intent is split into smaller chunks. Similarly, crucial information about the user experience may be lost if the artifacts are not able to carry the message leading the whole experience to be inevitably broken into unconnected parts.

The implemented product is different from the other artifacts in a sense that while the intermediate artifacts are a means to an end to implement the product, the product itself is the true representation of the design intent. The product, however, is similar to the intermediate artifacts in that it has instrumental value to the user experience—it is not the experience, it is designed for experience.

PRESERVING DESIGN INTENT. The interesting question to ask is, how to preserve the intention throughout the software development process and, more broadly, throughout the system's lifecycle. In the scope of user-centered software projects or in projects where UX is a goal, such as in the studied case, the work begins with an intention towards good UX (good being a mutual agreement rather than a measurable variable). It looks like many products in this context start with an idea of a good user experience but, though functional, the final product may turn out to be something entirely different.

In terms of quality attributes, the system may excel in many areas of quality but only indirectly and insignificantly contribute to the user experience if they are not grounded to the users' needs nor shared and evaluated. Therefore the non-incremental, non-technical UX activities are done together with the real users and as close to them as deemed necessary based on the software development lifecycle phase and the goals shared by the team. This establishes an ongoing bi-directional communication between the user and the software development team via the artifacts.

User experience is an intersubjective term, that is, it is widely recognized by most professionals. As a repercussion, the contextual meaning of the term and the causal explanation of why the system's UX is the way it is depends on who is asked if there are no proper measures and externalizations of the user experience. Important goals of the software system should be explicitly defined and measured.

***

Would a project benefit from a shared understanding of a system's quality attributes and their quantitative measures? Would it reduce the discrepancy between the developers and designers crafting an unfinished product and the customer expecting a finished and polished product? This could be beneficial in how the status of the progress is

communicated and how it is made more tangible. This study has found a correlation between user experience and system's quality attributes. Further research could be a context where a team collects quantitative data about the quality attributes to determine causality between the system and the user experience, for example, with Goal Question Metric (GQM), Key Performance Indicators (KPI) or Objective Key Results (OKR).

References

Agile Alliance. (2001). *Manifesto for Agile Software Development*. http://www.agilemanifesto.org/

Bargas-Avila, J. A., & Hornbæk, K. (2011). Old Wine in New Bottles or Novel Challenges: A Critical Analysis of Empirical Studies of User Experience. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2689–2698. https://doi.org/10.1145/1978942.1979336

Benyon, D. (2013). *Designing Interactive Systems: A Comprehensive Guide to HCI, UX & Interaction Design, 3rd ed.* (Comprehensive edition). Trans-Atlantic Publications, Inc.

Beyer, H., & Holtzblatt, K. (1998). *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufmann.

Beyer, H., Holtzblatt, K., & Baker, L. (2004). An Agile Customer-Centered Method: Rapid Contextual Design. In C. Zannier, H. Erdogmus, & L. Lindstrom (Eds.), *Extreme Programming and Agile Methods—XP/Agile Universe 2004* (pp. 50–59). Springer. https://doi.org/10.1007/978-3-540-27777-4_6

Bock, L. (2015). *Work Rules!: Insights from Inside Google That Will Transform How You Live and Lead* (1 edition). Twelve.

Boehm, B. W. (1984). Verifying and Validating Software Requirements and Design Specifications. *IEEE Software*, *1*(1), 75–88. https://doi.org/10.1109/MS.1984.233702

Boehm, B. W. (1991). Software Risk Management: Principles and Practices. *IEEE Software*, *8*(1), 32–41. https://doi.org/10.1109/52.62930

Brooks, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering* (Anniversary ed). Addison-Wesley Pub. Co.

Brown, R. (2001). *Group Processes Second Edition* (2nd edition). Wiley-Blackwell.

Charmaz, K. (2014). *Constructing Grounded Theory* (Second edition). SAGE Publications Ltd.

Christensen, C. M. (2016). *The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail* (Reprint edition). Harvard Business Review Press.

Conway, M. E. (1968). How Do Committees Invent. *Datamation*, *14*(4), 28–31.

Cooper, A., Reimann, R., Cronin, D., & Noessel, C. (2014). *About Face: The Essentials of Interaction Design* (4th ed.). Wiley Publishing.

Creswell, J. W., & Miller, D. L. (2000). Determining Validity in Qualitative Inquiry. *Theory Into Practice*, *39*(3), 124–130.

https://doi.org/10.1207/s15430421tip3903_2

Crosby, P. B. (1983). *Quality is free: The art of making quality certain*. New American Library.

Cross, N. (2011). *Design Thinking: Understanding How Designers Think and Work*. Berg Publishers.

DeMarco, T., & Lister, T. (2013). *Peopleware: Productive Projects and Teams* (3 edition). Addison-Wesley Professional.

Dhandapani, S. (2015). Agile Software Engineering in UCD: Literature Review. *2015 International Conference on Data and Software Engineering (ICoDSE)*, 37–41. https://doi.org/10.1109/ICODSE.2015.7436968

Fagerholm, F. (2015). *Software Developer Experience: Case Studies in Lean-Agile and Open Source Environments*. https://helda.helsinki.fi/handle/10138/158080

Fairbanks, G. H. (2010). *Just Enough Software Architecture: A Risk-Driven Approach* (1 edition). Marshall & Brainerd.

Ferreira, J., Sharp, H., & Robinson, H. (2012). Agile Development and User Experience Design Integration as an Ongoing Achievement in Practice. *2012 Agile Conference*, 11–20. https://doi.org/10.1109/Agile.2012.33

Ferreira, J., Sharp, H., & Robinson, H. (2010). Values and Assumptions Shaping Agile Development and User Experience Design in Practice. In A. Sillitti, A. Martin, X. Wang, & E. Whitworth (Eds.), *Agile Processes in Software Engineering and Extreme Programming* (pp. 178–183). Springer. https://doi.org/10.1007/978-3-642-13054-0_15

Flyvbjerg, B. (2016). Five Misunderstandings About Case-Study Research: *Qualitative Inquiry*. https://doi.org/10.1177/1077800405284363

Fox, D., Sillito, J., & Maurer, F. (2008). Agile Methods and User-Centered Design: How These Two Methodologies are Being Successfully Integrated in Industry. *Agile 2008 Conference*, 63–72. https://doi.org/10.1109/Agile.2008.78

Glaser, B. G., & Strauss, A. L. (1967). *The Discovery of Grounded Theory: Strategies for Qualitative Research*.

Gothelf, J., & Seiden, J. (2016). *Lean UX: Designing Great Products with Agile Teams* (2 edition). O'Reilly Media.

Hassenzahl, M., & Carroll, J. (2010). *Experience Design: Technology for All the Right Reasons*. Morgan and Claypool Publishers.

Heath, C., & Heath, D. (2007). *Made to Stick: Why Some Ideas Survive and Others Die* (1st edition). Random House.

Helkama, K. (2015). *Johdatus sosiaalipsykologiaan* (10., uudistettu painos). Edita.

Hokkanen, L., Kuusinen, K., & Väänänen, K. (2015). Early Product Design in Startups: Towards a UX Strategy. In P. Abrahamsson, L. Corral, M. Oivo, & B. Russo

(Eds.), *Product-Focused Software Process Improvement* (pp. 217–224). Springer International Publishing. https://doi.org/10.1007/978-3-319-26844-6_16

Holtzblatt, K. (2016). *Contextual Design: Design for Life* (2nd edn). Elsevier.

International Organization for Standardization. (2019). *Ergonomics of human-system interaction—Part 210: Human-centred design for interactive systems* (ISO Standard No. 9241-210:2019). https://www.iso.org/standard/77520.html

Kuusinen, K. (2015). *Integrating UX Work in Agile Enterprise Software Development*. https://doi.org/10.13140/RG.2.1.3372.9364

Kuusinen, K., Mikkonen, T., & Pakarinen, S. (2012). Agile User Experience Development in a Large Software Organization: Good Expertise but Limited Impact. In M. Winckler, P. Forbrig, & R. Bernhaupt (Eds.), *Human-Centered Software Engineering* (pp. 94–111). Springer. https://doi.org/10.1007/978-3-642-34347-6_6

Lane, R., Stanton, N. A., & Harrison, D. (2006). Applying hierarchical task analysis to medication administration errors. *Applied Ergonomics*, *37*(5), 669–679. https://doi.org/10.1016/j.apergo.2005.08.001

Larusdottir, M., Gulliksen, J., & Cajander, Å. (2017). A license to kill – Improving UCSD in Agile development. *Journal of Systems and Software*, *123*, 214–222. https://doi.org/10.1016/j.jss.2016.01.024

Law, E. L.-C., Roto, V., Hassenzahl, M., Vermeeren, A. P. O. S., & Kort, J. (2009). Understanding, Scoping and Defining User Experience: A Survey Approach. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 719–728. https://doi.org/10.1145/1518701.1518813

Lehman, M. M. (1979). On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle. *Journal of Systems and Software*, *1*, 213–221. https://doi.org/10.1016/0164-1212(79)90022-0

Lim, Y.-K., Stolterman, E., & Tenenberg, J. (2008). The Anatomy of Prototypes: Prototypes As Filters, Prototypes As Manifestations of Design Ideas. *ACM Trans. Comput.-Hum. Interact.*, *15*(2), 7:1–7:27. https://doi.org/10.1145/1375761.1375762

Magües, D. A., Castro, J. W., & Acuña, S. T. (2016). Usability in Agile Development: A Systematic Mapping Study. *2016 XLII Latin American Computing Conference (CLEI)*, 1–8. https://doi.org/10.1109/CLEI.2016.7833347

Mistrík, I., Grundy, J., Hoek, A. van der, & Whitehead, J. (2010). *Collaborative Software Engineering*. Springer Science & Business Media.

Norman, D. A. (1994). *Things That Make Us Smart: Defending Human Attributes In The Age Of The Machine* (Reprint edition). Basic Books.

Norman, D. A. (1998). *The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is so Complex, and Information Appliances Are the*

Solution. MIT Press.

Norman, D. A. (2013). *The Design of Everyday Things* (Revised and expanded edition). Basic Books.

Olson, J. S., & Kellogg, W. A. (Eds.). (2014). *Ways of Knowing in HCI* (2014 edition). Springer.

Papanek, V. J. (1985). *Design for the real world: Human ecology and social change* (2nd ed., completely rev). Academy Chicago.

Patton, J. (2014). *User Story Mapping: Discover the Whole Story, Build the Right Product* (P. Economy, Ed.; 1 edition). O'Reilly Media.

Patton, J. (2002). Hitting the Target: Adding Interaction Design to Agile Software Development. *OOPSLA 2002 Practitioners Reports*, 1–ff. https://doi.org/10.1145/604251.604255

Patton, M. Q. (2001). *Qualitative Research & Evaluation Methods* (3rd edition). SAGE Publications, Inc.

Ralph, P. (2015). The Sensemaking-Coevolution-Implementation Theory of software design. *Science of Computer Programming*, *101*, 21–41. https://doi.org/10.1016/j.scico.2014.11.007

Ralph, P., & Narros, J. E. (2013). Complexity, process and agility in small development teams: An exploratory case study. *Proceedings - Pacific Asia Conference on Information Systems, PACIS 2013*.

Ries, E. (2011). *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses* (1 edition). Currency.

Runeson, P., & Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, *14*(2), 131. https://doi.org/10.1007/s10664-008-9102-8

Runeson, P., Host, M., Rainer, A., & Regnell, B. (2012). *Case Study Research in Software Engineering: Guidelines and Examples* (1 edition). Wiley.

Salah, D., Paige, R., & Cairns, P. (2014). Integrating Agile Development Processes and User Centred Design- A Place for Usability Maturity Models? In S. Sauer, C. Bogdan, P. Forbrig, R. Bernhaupt, & M. Winckler (Eds.), *Human-Centered Software Engineering* (pp. 108–125). Springer. https://doi.org/10.1007/978-3-662-44811-3_7

Schön, D. A. (1984). *The Reflective Practitioner: How Professionals Think In Action* (1 edition). Basic Books.

Seaman, C. B. (1999). Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Trans. Software Eng.*, *25*, 557–572. https://doi.org/10.1109/32.799955

Shadish, W. R., Cook, T. D., & Campbell, D. T. (2001). *Experimental and*

Quasi-Experimental Designs for Generalized Causal Inference (2 edition). Cengage Learning.

Sohaib, O., & Khan, K. (2010). Integrating Usability Engineering and Agile Software Development: A Literature Review. *2010 International Conference On Computer Design and Applications*, *2*, V2-32-V2-38. https://doi.org/10.1109/ICCDA.2010.5540916

Stanton, N. A. (2006). Hierarchical task analysis: Developments, applications, and extensions. *Applied Ergonomics*, *37*(1), 55–79. https://doi.org/10.1016/j.apergo.2005.06.003

Stol, K.-J., Ralph, P., & Fitzgerald, B. (2016). Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. *Proceedings of the 38th International Conference on Software Engineering*, 120–131. https://doi.org/10.1145/2884781.2884833

Sy, D. (2007). Adapting Usability Investigations for Agile User-Centered Design. *Journal of Usability Studies*, *2*(3), 112–132.

Whitehead, J. (2007). Collaboration in Software Engineering: A Roadmap. *Future of Software Engineering (FOSE '07)*, 214–225. https://doi.org/10.1109/FOSE.2007.4

Yin, R. K. (2017). *Case Study Research and Applications: Design and Methods* (Sixth edition). SAGE Publications, Inc.